

1. Gestión de la información almacenada en Ficheros.

Índice de contenidos

- 1.1. CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE FICHEROS Y DIRECTORIOS.
- 1.2. FLUJOS.
- 1.3. FORMAS DE ACCESO A UN FICHERO.
- 1.4. CLASES PARA GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS.
- 1.5. TRABAJO CON FICHEROS XML
- 1.6. LIBRERÍAS PARA CONVERSIÓN DE DOCUMENTOS XML A OTROS FORMATOS.

1.1. CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE FICHEROS Y DIRECTORIOS,

Muchos conceptos que vamos a ver en esta unidad y en las siguientes unidades, van a hacer hincapié en temas que probablemente hayas estudiado de programación. Esto te va a facilitar el hecho de que los conceptos que vamos a tratar ya te sean conocidos. Veremos CREACIÓN, BORRADO, COPIA, MOVIMIENTO, ENTRE OTRAS.

Cuando guardamos datos que a su vez se guardan en ficheros se les denominan, **datos persistentes**, porque persisten más allá de la ejecución de la aplicación que los trata. Los ordenadores almacenan los ficheros tanto en unidades de almacenamiento secundario como en discos duros, pendrives, discos duros portátiles, o en la nube. En esta unidad veremos, entre otras cosas, cómo hacer con Java las operaciones de crear, actualizar y procesar ficheros.

Definición

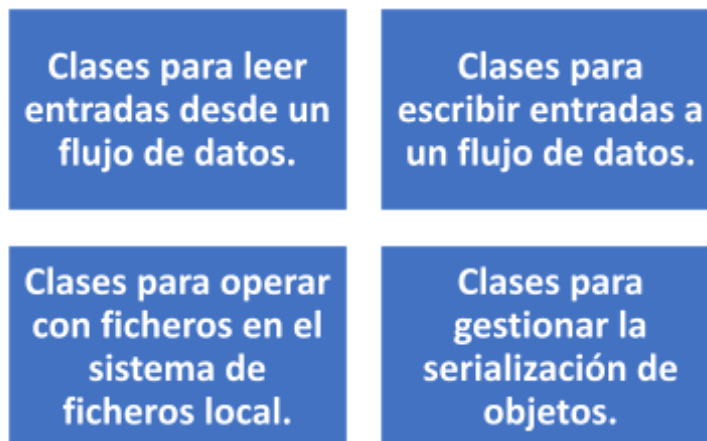
A las operaciones, que constituyen un flujo de información del programa con el exterior, se les conoce como **Entrada/Salida (E/S)**.

Nota

Las operaciones de E/S en Java las proporciona el paquete estándar de la API de Java denominado `java.io` que incorpora interfaces, clases y excepciones para acceder a todo tipo de ficheros.

Librería `java.io`

Esta librería contiene las clases necesarias para gestionar las operaciones de entrada y salida con Java. Estas clases de E/S se agrupan de la siguiente manera:



Clases E/S en Java

Clase File

La **clase File** sirve para realizar operaciones con ficheros, así mismo nos permite filtrar ficheros, o sea, así como obtener aquellos con una característica determinada.

En realidad, esta clase lo que hace es proporcionarnos una **representación abstracta de ficheros** y directorios. Lo cual permite que podamos examinar y manipular archivos y directorios, al margen del sistema operativo que usemos.

Al instanciar la clase File, estas representarán nombres de archivo, no los archivos en sí mismos.

Importante

Podría ocurrir que dicho archivo, no corresponda a un nombre que exista, por lo que algo fundamental, es controlar las excepciones que nos podamos encontrar.

¿Qué es lo que hace el objeto de clase File?

permite examinar el nombre del archivo, descomponerlo en su rama de directorios o crear el archivo si no existe mediante el método:

Veamos un ejemplo sencillo:

Ver las propiedades de un fichero:

```
package ficheros;
```

```
import java.io.*;
public class VerInformacionFicheros1 {
public static void main(String[] args) {
    System.out.println("INFORMACIÓN SOBRE EL DIRECTORIO:");
    File direc = new File(".\\src\\ficheros\\");
    if(direc.exists()){
        System.out.println("Nombre del directorio : "+direc.getName());
        System.out.println("Ruta : "+direc.getPath());
        System.out.println("Ruta absoluta : "+direc.getAbsolutePath());
        System.out.println("Lectura : "+direc.canRead());
        System.out.println("Escritura : "+direc.canWrite());
    }
}
```

```

        System.out.println("Tamaño                : "+direc.length()+ " Kb"); //
El tamaño es expresado en bytes
        System.out.println("Directorio         : "+direc.isDirectory());
        System.out.println("Fichero          : "+direc.isFile());
        System.out.println("Nombre del directorio padre:
"+direc.getParent());
    }
}
}
}

```

Nos devolvería esto:

```

terminated> verinformacionficheros java Application C:\Program Files\Java\jdk-19\bin\javaw.exe (20 Oct 2023, 13:57:03 - 13:57:03) [pid: 16720]

```

INFORMACIÓN SOBRE EL FICHERO:

```

Nombre del directorio : ficheros
Ruta                  : .\src\ficheros
Ruta absoluta         : C:\Users\alber\eclipse-workspace\Cenec\. \src\ficheros
Lectura              : true
Escritura             : true
Tamaño                : 4096 Kb
Directorio            : true
Fichero               : false
Nombre del directorio padre: .\src

```

Método	Descripción
File(String path)	Constructor de File que crea una instancia de File con la ruta especificada.
exists()	Comprueba si el archivo o directorio especificado existe.
getName()	Obtiene el nombre del archivo o directorio sin la ruta completa.
getPath()	Obtiene la ruta del archivo o directorio especificado.
getAbsolutePath()	Obtiene la ruta absoluta del archivo o directorio especificado.
canRead()	Comprueba si se puede leer el archivo o directorio.
canWrite()	Comprueba si se puede escribir en el archivo o directorio.
length()	Obtiene el tamaño del archivo en bytes.
isDirectory()	Comprueba si el objeto File representa un directorio.
isFile()	Comprueba si el objeto File representa un archivo.
getParent()	Obtiene la ruta del directorio padre del archivo o directorio especificado.

_En el caso de un directorio, solo tendríamos que cambiar

```
File direc = new File(".\\src\\ficheros\\");
```

```
createNewFile();
```

En el caso de que dichos archivos existan, a través del objeto file, un programa podríamos: **Examinar los atributos del archivo. Cambiar su nombre. Borrarlo. Cambiar sus permisos.**

Veamos cómo podríamos crear un directorio:

Usando **mkdir** podemos crear la carpeta.

```

package ficheros;

import java.io.File;

public class CrearDirectorio3 {
    public static void main(String[] args) {
        String ruta=".\\src\\ficheros\\";
        String carpeta = "ejercicio1";
        File directorio = new File(ruta+carpeta);
        if (directorio.exists()==true) {
            System.out.println("la carpeta existe idiota");
        } else {
            directorio.mkdir();
            System.out.println("carpeta creada");
        }
    }
}

```

Ahora probamos para crear un fichero en dicha carpeta:

Usando createNewFile()

```

package ficheros;

import java.io.File;
import java.io.IOException;

public class CrearFicheroEnDirectorio {
    public static void main(String[] args) {
        String ruta = ".\\src\\ficheros\\ejercicio1";
        String nombreFichero = "fichero1.txt"; // Nombre del fichero que deseas crear

        // Combinar la ruta del directorio con el nombre del fichero
        String rutaFichero = ruta + File.separator + nombreFichero;

        File fichero = new File(rutaFichero);

        if (fichero.exists()) {
            System.out.println("El fichero ya existe.");
        } else {
            try {
                boolean creado = fichero.createNewFile();
                if (creado) {
                    System.out.println("El fichero ha sido creado con éxito.");
                } else {
                    System.out.println("No se pudo crear el fichero.");
                }
            } catch (IOException e) {
                System.out.println("Error al crear el fichero: " + e.getMessage());
            }
        }
    }
}

```

1º Borrar un fichero

Para borrar un fichero podemos usar la clase File, comprobando previamente si existe el fichero, del siguiente modo: Para borrar el fichero deberemos de invocar el método `.delete()` de la clase File

Sabias que

Desde este enlace, <http://www.java2s.com/Code/Jar/o/Downloadorgapachecommonsiojar.htm> podemos descargarnos el paquete `org.apache.commons.io`; que nos va a permitir usar `la clase FileUtils` y el método `FileUtils.deleteDirectory`;

Ejemplo

Borrar el fichero 1 de la carpeta fichero que está en src:

```
public class BorrarFicheros6 {
    public static void main(String[] args) {
        String ruta = ".\\src\\ficheros\\ejercicio1";
        String nombreFichero = "fichero1.txt"; // Nombre del fichero que deseas
        borrar

        // Combinar la ruta del directorio con el nombre del fichero
        String rutaFichero = ruta + File.separator + nombreFichero;

        File fichero = new File(rutaFichero);

        if (fichero.exists()) {
            boolean borrado = fichero.delete();
            if (borrado) {
                System.out.println("El fichero ha sido borrado con éxito.");
            } else {
                System.out.println("No se pudo borrar el fichero.");
            }
        } else {
            System.out.println("El fichero no existe en la ubicación
            especificada.");
        }
    }
}
```

2º Borrar un directorio

En Java si queremos borrar un directorio con Java, tendremos que borrar cada uno de los ficheros y directorios que éste contenga.

```
File[] ficheros = directorio.listFiles();
```

Si el elemento es un directorio, podemos saberlo utilizando el método `isDirectory()`.

En este ejemplo vamos a borrar la carpeta fichero que está en src

```
package ficheros;

import org.apache.commons.io.FileUtils;
import java.io.File;
import java.io.IOException;

public class BorrarCarpeta7 {
    public static void main(String[] args) throws IOException {
        // Especifica la ruta de la carpeta que quieres borrar
        File carpeta = new File("/ruta/a/la/carpeta");

        // Borra la carpeta y todos sus archivos y subdirectorios
        if (carpeta.exists() && carpeta.isDirectory()) {
            System.out.println("El directorio existe.");
            FileUtils.deleteDirectory(carpeta);
            System.out.println("El directorio ha sido borrado.");
        } else {
            System.out.println("El directorio no existe o no es un directorio válido.");
        }
    }
}
```

1.2. FLUJOS.

En este punto vamos a ver el tema de los flujos, los flujos basados en bytes y los flujos basados en caracteres, pero primero veamos en qué consisten los flujos.

Definición

Un flujo de datos en **java** es lo que permite que un programa realizado en **Java** u otro lenguaje de programación pueda leer o escribir datos en otro periférico, programa, etc. Un flujo es una abstracción de todo aquello que produce o consume información. Un flujo es una abstracción de todo aquello que produce o consume información. En Java la vinculación de este flujo al dispositivo físico la hace el sistema de entrada y salida de Java

Ejemplo

En el caso de Java, realizamos este flujo de entrada o salida de al que denominamos E/S a través de un flujo, o también denominado Stream, Las clases y métodos de E/S que empleamos tienen la particularidad de ser independiente del dispositivo con el que estemos actuando.

Dentro del paquete java.io, nos encontramos dos tipos de flujos:

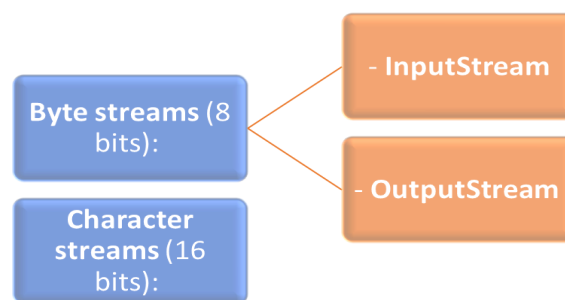
- **Byte streams (8 bits):**

orientado a la lectura y escritura de datos binarios, proporciona lo necesario para la gestión de entradas y salidas de bytes. El tratamiento del flujo de bytes viene determinado por dos clases abstractas que son:

- **InputStream**
- **OutputStream**

- **Character streams (16 bits):**

Los flujos de caracteres están determinados por dos clases abstractas, en este caso: Reader y Writer. Dichas clases manejan flujos de caracteres Unicode.



Flujos en Java

nota

Tanto en el flujo Byte streams como en el caso de Character streams, también de ellas derivan subclases concretas que implementan los métodos definidos en ellas siendo los más destacados los métodos **read()** y **write()** que leen y escriben caracteres de datos respectivamente.

1.2.1. Flujos basados en bytes.

Para el tratamiento de los flujos de bytes, hemos dicho que Java tiene dos clases abstractas que hemos mencionado antes, son **InputStream** y **OutputStream**.

importante

Un **archivo binario** es un **archivo** informático que contiene información de cualquier tipo codificada en **binario** para el propósito de almacenamiento y procesamiento en ordenadores. Por ejemplo **los archivos** informáticos que almacenan texto formateado o fotografías, así como **los archivos** ejecutables que contienen programas.

En esta tabla podemos ver las clases principales que heredan de **OutputStream**, para la escritura de ficheros binarios son:

FileOutputStream	ObjectOutputStream	DataOutputStream
<p>Escribe bytes en un fichero. En el caso de que el archivo existe, cuando vayamos a escribir sobre él, se borrará.</p> <p>Para añadir los datos al final de éste, habrá que usar el constructor FileOutputStream(String filePath, boolean append), poniendo append a true.</p>	<p>Convierte objetos y variables en vectores de bytes que pueden ser escritos en un OutputStream.</p>	<p>Da formato a los tipos primitivos y objetos String, convirtiéndolos en un flujo de forma que cualquier DataInputStream, de cualquier máquina, los pueda leer. Todos los métodos empiezan por "write", como writeByte(), writefloat(), etc.</p>

Importante

InputStream, para lectura de ficheros binarios, destacamos:

- FileInputStream: lee bytes de un fichero.
- ObjectInputStream: convierte en objetos y variables los vectores de bytes leídos de un InputStream

Ejemplo

Vamos a ver en este ejemplo, como podemos crear un flujo de salida y otro de entrada a un archivo binario mediante las clase **FileInputStream** y **FileOutputStream**:

```

1 package streamsBinarios;
2
3 import java.io.*;
4 public class WriteFileBytes {
5     public static void main(String[] args) throws IOException {
6         File fichero = new File(".\\src\\fileBytes.dat");//declara el fichero
7         //crea flujo de salida hacia el fichero, si no existe el fichero se crea
8         // al poner a true el segundo parámetro indicamos que se añada la información al final del archivo
9         // para no sobrescribir
10        FileOutputStream fileout = new FileOutputStream(fichero,true);
11        //crea flujo de entrada, en caso de no existir el fichero dará un FileNotFoundException
12        FileInputStream filein = new FileInputStream(fichero);
13
14        for (int i=1; i<100; i++)
15            fileout.write(i); //escribe los datos en el flujo de salida byte a byte
16
17        fileout.close(); //cierra el stream de salida para liberar los recursos
18
19        //visualizar los datos del fichero
20        while (filein.available() != 0) // comprobamos que haya bytes restantes por leer
21            System.out.println(filein.read()); //lee los datos del flujo de entrada byte a byte
22
23        filein.close(); //cierra stream de entrada para liberar recursos
24    }
25 }
26

```

Objetos en ficheros binarios.

Para guardar objetos en ficheros binarios; se dispone de una serie de métodos con los que podremos guardar y leer objetos en ficheros binarios:

Para leer un objeto del **ObjectInputStream**

- **readObject()** throws **IOException**, **ClassNotFoundException**.

para escribir el objeto especificado en el **ObjectOutputStream**

- **void writeObject(Object obj)** throws **IOException**.

Importante

La serialización de objetos consiste en generar una secuencia de bytes lista para su almacenamiento o transmisión. **Después**, mediante la deserialización, el estado original del **objeto** se puede reconstruir. **como serializable**, sin que tengamos que implementar ningún método). import java.

la clase ObjectOutputStream y ObjectInputStream

Para leer y escribir objetos serializables a un stream se utilizan estas clases. Veamos el siguiente ejemplo en donde ponemos en funcionamiento una clase denominada empleado.

Ejemplo

```
1 package objetosBinarios;
2
3 import java.io.Serializable;
4
5 public class Empleado implements Serializable{
6
7     private static final long serialVersionUID = 7833717614651250909L;
8     private String nombre;
9     private int antiguedad;
10
11     public Empleado(String nombre,int antiguedad){
12         this.nombre=nombre;
13         this.antiguedad=antiguedad;
14     }
15
16     public Empleado(){
17         this.nombre="Pedro";
18         this.antiguedad = 0;
19     }
20
21     public void setNombre(String nombre){
22         this.nombre=nombre;
23     }
24
25     public void setAntiguedad(int antiguedad){
26         this.antiguedad=antiguedad;
27     }
28
29     public String getNombre(){
30         return nombre;
31     }
32
33     public int getAntiguedad(){
34         return antiguedad;
35     }
36 }
37
38
39
```

Ejemplo

En este caso, tenemos que leer objetos empleado del fichero necesitamos el flujo de entrada a disco FileInputStream y a continuación crear el flujo de entrada ObjectInputStream que es el que procesa los datos y se ha de vincular al fichero de FileInputStream:

```

1 package objetosBinarios;
2
3 import java.io.*;
4
5 public class ReadFichObject {
6     public static void main(String[] args) throws IOException, ClassNotFoundException {
7         Empleado empleado; // definir la variable empleado
8         File fichero = new File ("..\\src\\fichempleado.dat");
9         ObjectInputStream dataIN = new ObjectInputStream(new FileInputStream(fichero));
10
11         int i = 1;
12         try {
13             while (true) { // lectura del fichero
14                 empleado = (Empleado) dataIN.readObject(); // leer un empleado
15                 System.out.println(i + " -> " + empleado.getNombre() + ", " + empleado.getAntiguedad());
16                 i++;
17             }
18         } catch (EOFException eo) { // se produce esta excepcion al llegar al final del fichero
19             System.out.println("FIN DE LECTURA.");
20         } catch (StreamCorruptedException x) {
21             x.printStackTrace();
22         }
23
24         dataIN.close(); // cerrar stream de entrada
25     }
26 }
27
28

```

2.2.- Flujos basados en caracteres.

Los flujos de de caracteres en Java dispone de dos clases abstractas: **Reader** y **Writer**.

Nota

Si se usan sólo `FileInputStream`, `FileOutputStream`, `FileReader` o `FileWriter`, cada vez que se efectúa una lectura o escritura, se hace físicamente en el disco duro. Si se leen o escriben pocos caracteres cada vez, el proceso se hace costoso y lento por los muchos accesos al disco duro.

Para evitar precisamente ese problema, tenemos una serie de clases que son:

- `BufferedReader`
- `BufferedInputStream`
- `BufferedWriter`
- `BufferedOutputStream`

Esto permite crear una transición entre **java** y el archivo. La información del archivo se volcará en el **buffer** y **java** accede al buffer para cargar o descargar la información poco a poco.

En el siguiente ejemplo podremos ver el funcionamiento de los flujos basados en caracteres.

:

Ejemplo

En este ejemplo. Se llega al final del fichero cuando el método `read()` devuelve -1.

```

1 package streamsTexto;
2
3 import java.io.*;
4
5 public class readFichTexto {
6     public static void main(String[] args) throws IOException {
7         File fichero = new File("C:\\prueba\\myfile.txt"); //declara el fichero
8         FileReader fiche = new FileReader(fichero); //crea el flujo de entrada
9
10        int i;
11        while ((i = fiche.read()) != -1) //se lee el archivo hasta que encuentre el carácter -1
12            System.out.print((char) i); //cast a char
13
14        //para leer de 20 en 20 utilizamos el metodo read pasándole el buffer de caracteres
15        /* char b[] = new char [20];
16         while ((i = fiche.read(b)) != -1)
17             System.out.println(b);*/
18
19        //System.out.println( (char) i + "=="> i);
20
21        fiche.close(); //cerrar fichero
22    }
23 }
24 }
25

```

Búferes en ficheros de texto

Un aspecto interesante de `BufferedReader` es que permite una línea del fichero y la devuelve, utilizando para ello el método `readLine()` o en todo caso devuelve null si no hay nada que leer o llegamos al final del fichero. Otro método es `read()` para leer un carácter.

Para construir un `BufferedReader` necesitamos la clase `FileReader`.

```
BufferedReader fichero = new BufferedReader(new FileReader(NombreFichero));
```

Ejemplo

El siguiente ejemplo lee el fichero `readFichTexto.java` línea por línea y las va visualizando en pantalla, en este caso las instrucciones se han agrupado dentro de un bloque try-catch:

```

1 package file;
2
3 import java.io.*;
4 public class readFichTextoBuf {
5     public static void main(String[] args) {
6         try{
7             File fic = new File("C:\\prueba\\myfile.txt");//declara fichero
8             BufferedReader fichero = new BufferedReader(new FileReader(fic));
9
10            String linea;
11            while((linea = fichero.readLine())!=null)
12                System.out.println(linea);
13            fichero.close();
14        }
15        catch (FileNotFoundException fn ){
16            System.out.println("No se encuentra el fichero");}
17        catch (IOException io) {
18            System.out.println("Error de E/S ");}
19    }
20 }
21 }
22

```

<

Consola × Problems Intérprete de depuración Structure Coverage

<terminado> readFichTextoBuf [Aplicación Java] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe (13 ene 2022 18:50)

hola
esto
es
una prueba

La clase BufferedWriter

Deriva de la clase `Writer`. Esta clase añade un buffer para realizar una escritura eficiente de caracteres. Para construir un `BufferedWriter` necesitamos la clase `FileWriter`:

```
BufferedWriter fichero = new BufferedWriter (new FileWriter (NombreFichero));
```

Ejemplo

El siguiente ejemplo escribe 20 filas de caracteres en un fichero de texto y después de escribir cada fila salta una línea con el método `newLine()`:

```
1 package streamsTexto;
2
3 import java.io.*;
4 public class writerFichTextoBuf {
5     public static void main(String[] args) {
6         try{
7             BufferedWriter fichero = new BufferedWriter (new FileWriter("C:\\prueba\\myfile.txt"));
8             for (int i=1; i<21; i++){
9                 fichero.write("Fila numero: "+i); //escribe una línea
10                fichero.newLine(); //escribe un salto de línea
11            }
12            fichero.close();
13        }
14        catch (FileNotFoundException fn ){
15            System.out.println("No se encuentra el fichero");}
16        catch (IOException io) {
17            System.out.println("Error de E/S ");}
18    }
19 }
20
21
```

Resultado

```
Fila numero: 1
Fila numero: 2
Fila numero: 3
Fila numero: 4
Fila numero: 5
Fila numero: 6
Fila numero: 7
Fila numero: 8
Fila numero: 9
Fila numero: 10
Fila numero: 11
Fila numero: 12
Fila numero: 13
Fila numero: 14
Fila numero: 15
Fila numero: 16
Fila numero: 17
Fila numero: 18
Fila numero: 19
Fila numero: 20
```

La clase PrintWriter

Proviene de `Writer`, posee los métodos `print(String)` y `println(String)` (idénticos a los de `System.out`) para escribir en un fichero. Ambos reciben un `String` y lo imprimen en un fichero, el segundo método salta de línea. Para construir un `PrintWriter` necesitamos la clase `FileWriter`:

```
PrintWriter fichero = new PrintWriter (new FileWriter(NombreFichero));
```

Ejemplo

```
1 package streamsTexto;
2
3 import java.io.*;
4 public class writeFichTextoPrint {
5     public static void main(String[] args) {
6         try{
7             PrintWriter fichero = new PrintWriter (new FileWriter("C:\\prueba\\myfile.txt"));
8             for (int i=1; i<=11; i++){
9                 fichero.println("Fila número: "+i); //escribe una línea
10            }
11            fichero.close();
12        }
13        catch (FileNotFoundException fn ){
14            System.out.println("No se encuentra el fichero");}
15        catch (IOException io) {
16            System.out.println("Error de E/S ");}
17    }
18 }
19 }
```

myfile: Bloc de notas

Archivo Edición Formato Ver Ayuda

Fila número: 1
Fila número: 2
Fila número: 3
Fila número: 4
Fila número: 5
Fila número: 6
Fila número: 7
Fila número: 8
Fila número: 9
Fila número: 10

Recurso web

En este artículo verás diferentes ejemplos sobre el tratamiento del buffer en Java

[StringBuffer, StringBuilder Java. Ejemplo. Diferencias entre clases. Criterios para elegir. Métodos. \(CU00914C\) \(aprenderaprogramar.com\)](#)

Vídeo

En este video se trata el tema de las - Escritura de archivos en Java(Clases: FileWriter y PrintWriter).

<https://www.youtube.com/watch?v=kFR2XRfJxZw>

1.3. FORMAS DE ACCESO A UN FICHERO.

En el apartado anterior, hemos estudiado **dos tipos de ficheros (de texto o binarios) y dos tipos de acceso a los ficheros (secuencial o aleatorio).**

Saber más

En Java IO, las canalizaciones se utilizan para proporcionar comunicación entre dos subprocesos en la misma JVM. Por tanto, la canalización también puede ser la fuente o el destino de los datos. En diferentes JVM (diferentes procesos), no puede usar tuberías para conectar subprocesos. El concepto de tuberías en Java es diferente de las tuberías de Linux o Unix. En Linux o Unix, las tuberías se pueden utilizar para conectar dos procesos que se ejecutan en diferentes espacios de direcciones. En Java, la parte de comunicación debe estar en el mismo proceso y se pueden utilizar diferentes subprocesos.

En esta imagen podemos ver los diferentes acceso:

Acceso aleatorio:

- los archivos de acceso aleatorio, al igual que lo que sucede usualmente con la memoria. Esto implica que el archivo debe estar disponible en su totalidad al momento de ser accedido.

Acceso secuencial:

- En este caso los datos se leen de manera secuencial, desde el comienzo del archivo hasta el final. Este es el caso de la lectura del teclado o la escritura en una consola de texto.

3.1. Operaciones sobre ficheros de acceso secuencial.

Las operaciones sobre ficheros de acceso secuencial más comunes son:

Crear un fichero o abrirlo para grabar datos.	Leer datos del fichero.
Borrar información de un fichero.	Copiar datos de un fichero a otro.
Búsqueda de información en un fichero.	Cerrar un fichero.

3.2.- Operaciones sobre ficheros de acceso aleatorio.

Esta es una manera de acceder a un fichero como si fuera una base de datos, de manera aleatoria, para ello. Java nos proporciona una clase **RandomAccessFile** para este tipo de entrada/salida.

Clase RandomAccessFile

Esta clase permite leer y escribir sobre el fichero. Necesita que le especifiquemos el modo de acceso al construir un objeto de esta clase: sólo lectura o bien lectura y escritura. Posee métodos específicos de desplazamiento como **seek(long posicion)** o **skipBytes(int desplazamiento)** para poder movernos de un registro a otro del fichero, o posicionarnos directamente en una posición concreta del fichero.

Por esas características que presenta la clase, un archivo de acceso directo tiene sus registros de un tamaño fijo o predeterminado de antemano.

La clase posee dos constructores:

En el primer caso se pasa un objeto `File` como primer parámetro, mientras que en el segundo caso es un `String`. El modo es: "r" si se abre en modo lectura o "rw" si se abre en modo lectura y escritura.

Ejemplo

El ejemplo que se muestra a continuación inserta datos de empleados en un fichero aleatorio. Por cada empleado también se insertará un identificador que coincidirá con el índice +1 con el que se recorren los arrays. La longitud del registro de cada empleado es la misma (36 bytes) y los tipos que se insertan y su tamaño en bytes es el siguiente:

- Se inserta en primer lugar un entero, que es el identificador, ocupa 4 bytes.
- A continuación una cadena de 10 caracteres, es el apellido, cada carácter Unicode ocupa 2 bytes, por tanto el apellido ocupa 20 bytes.
- Un tipo entero que es el departamento, ocupa 4 bytes.
- Un tipo `Double` que es el salario, ocupa 8 bytes.

Tamaño de otros tipos: `short` (2 bytes), `byte` (1 byte), `long` (8 bytes), `boolean` (1bit), `float` (4 bytes), etc.

El fichero se abre en modo "rw" para lectura y escritura:

```
1 package accesoAleatorio;
2
3 import java.io.*;
4 public class EscribirFileAleatorio {
5     public static void main(String[] args) throws IOException {
6         File fichero = new File("C:\\prueba\\AleatorioEmple.dat");
7         //declara el fichero de acceso aleatorio
8         RandomAccessFile file = new RandomAccessFile(fichero, "rw");
9         //arrays con los datos
10        String apellido[] = {"RUIZ", "RODRIGUEZ", "LUCENA", "REINA",
11                             "MÁLAGA", "GÓMEZ", "PÉREZ"}; //apellidos
12        int dep[] = {10, 20, 20, 30, 20, 10, 20}; //departamentos
13        Double salario[] = {1200.45, 2200.60, 3400.0, 1800.56,
14                             2600.0, 1600.87, 2000.0}; //salarios
15
16        StringBuffer buffer = null; //buffer para almacenar apellido
17        int n = apellido.length; //número de elementos del array
18
19        for (int i=0; i<n; i++){ //recorremos los arrays
20            file.writeInt(i+1); //uso i+1 para identificar empleado
21            buffer = new StringBuffer(apellido[i]);
22            buffer.setLength(10); //10 caracteres para el apellido
23            file.writeChars(buffer.toString()); //insertar apellido
24            file.writeInt(dep[i]); //insertar departamento
25            file.writeDouble(salario[i]); //insertar salario
26        }
27        file.close(); //cerrar fichero
28    }
29 }
30
```

En este artículo puedes ver algunos ejemplos de la clase `RandomAccessFile`

<http://puntocomnoesunlenguaje.blogspot.com/2013/06/java-ficheros-acceso-aleatorio.html>

Vídeo

En este vídeo, puedes ampliar conocimientos sobre la clase `RandomAccessFile`

https://www.youtube.com/watch?v=gcPau_67V8s

1.4. CLASES PARA GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS.

Como hemos visto en los apartados anteriores, existen las siguientes clases asociadas a operaciones de gestión de ficheros y directorios.

- Clases asociadas a las operaciones de gestión de ficheros y directorios, creación, borrado, copia, movimiento, entre otras.
- Flujos. Flujos basados en bytes y flujos basados en caracteres.
- Formas de acceso a un fichero. Operaciones básicas sobre ficheros de acceso secuencial y aleatorio. Ventajas e inconvenientes de las distintas formas de acceso.
- Clases para gestión de flujos de datos desde/hacia ficheros.

La E/S en Java sigue el siguiente modelo:

**Abrir, usar, cerrar
flujo**

Flujos estándar:

- `System.in`
- `System.out`
- `System.err`

La E/S en Java

Dos tipos de clases de E/S:

**Readers y Writers
para texto**
• Basados en el tipo `char`

**Streams (`InputStream`
y `OutputStream`) para
datos binarios**
• Basados en el tipo `byte`

Dos tipos de clases de E/S:

Los flujos de E/S se pueden combinar para facilitar su uso.

Importante

La E/S suele ser propensa a errores, por tanto.

- Implica interacción con el entorno exterior

- Excepción: IOException

Recurso web

En este artículo puedes ampliar más información:

<https://codesitio.com/recursos-utiles-para-tu-web-o-blog/cursos/curso-de-java-flujos-de-datos-en-java/>

Vídeo

En este vídeo puedes ver más información sobre la gestión de ficheros en Java

<https://www.youtube.com/watch?v=OcvpB4O-2RU>

1.5. TRABAJO CON FICHEROS XML

Definición

XML es el acrónimo de Extensible Markup Language, es decir, es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos. El lenguaje XML proporciona una plataforma para definir elementos para crear un formato y generar un lenguaje personalizado. lo cual permite el intercambio de información y proporciona una visión estructural de la información.

las aplicaciones que utilizan este tipo de ficheros, deben atender a una serie de protocolos:

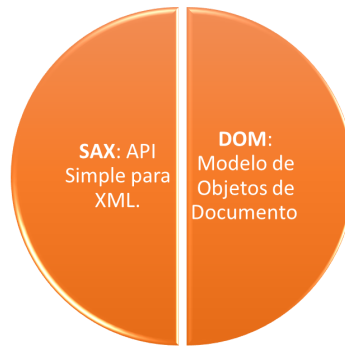
1º La posibilidad de leer ficheros textos codificado según estándar XML

2º Procesar estos datos para obtener unos resultados

Analizadores sintácticos(parser)

Un **parser** podría ser definido como un **programa** que analiza una porción de texto para determinar su estructura lógica: la fase de **parsing** en un compilador toma el texto de un **programa** y produce un árbol sintáctico que representa la estructura del **programa**. **XML parser** es una biblioteca de software o un paquete que proporciona la interfaz para aplicaciones cliente para trabajar con documentos **XML**. Se verifica la estructura del documento **XML** y también puede validar los documentos **XML**. . El objetivo de un **parser** es transformar **XML** en un código legible.

Algunos de los procesadores más empleados son:



procesadores

Importante

Un procesador que utilice este planteamiento(SAX) lee un fichero XML de forma secuencial y produce una secuencia de eventos (comienzo/fin del documento, comienzo/fin de una etiqueta, etcétera) en función de los resultados de la lectura. Cada evento invoca un método definido por el programador.

1.5.1.- Ficheros XML con DOM

Un procesador XML que utilice DOM almacena toda la estructura del documento en memoria en forma de árbol con nodos padre, nodos hijo y nodos finales (sin descendientes).

Nota

Para poder trabajar con DOM en Java necesitas las clases e interfaces que componen el paquete *org.w3c.dom* (contenido en el JSDK) y el paquete *javax.xml.parsers* del API estándar de Java que proporciona un par de clases abstractas que toda implementación DOM para Java debe extender.

Recurso web

puedes ver la ayuda de Oracle en este enlace:

<https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html>

Los programas Java que utilicen DOM necesitan estas interfaces:

Document.

Es un objeto que equivale a un ejemplar de un documento XML, Permite crear nuevos nodos en el documento.

Element.

Cada elemento del documento XML tiene un equivalente en un objeto de este tipo.

Node. Representa a cualquier nodo del documento.

NodeList. Contiene una lista con los nodos hijos de un nodo.

Attr.

Permite acceder a los atributos de un nodo.

Text.

Son los datos carácter de un elemento.

CharacterData. Representa a los datos carácter presentes en el documento.

DocumentType. Proporciona información contenida en la etiqueta <!DOCTYPE>.

¿Cómo creamos un fichero XML?

Primero para poder crear ficheros XML, necesitamos importar las siguientes librerías:

- `import org.w3c.dom.*;`
- `import javax.xml.parsers.*;`
- `import javax.xml.transform.*;`
- `import javax.xml.transform.dom.*;`
- `import javax.xml.transform.stream.*;`
- `import java.io.*;`

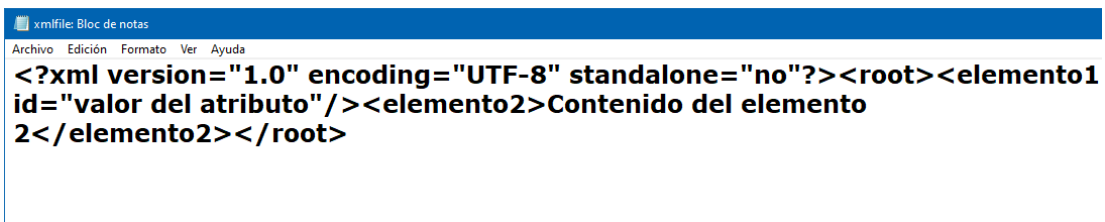
Veamos mejor un ejemplo, en este caso, hemos generado un fichero Xml, en la carpeta prueba.

```

1 import java.io.File;
2 import javax.xml.parsers.DocumentBuilder;
3 import javax.xml.parsers.DocumentBuilderFactory;
4 import javax.xml.parsers.ParserConfigurationException;
5 import javax.xml.transform.Transformer;
6 import javax.xml.transform.TransformerException;
7 import javax.xml.transform.TransformerFactory;
8 import javax.xml.transform.dom.DOMSource;
9 import javax.xml.transform.stream.StreamResult;
10 import org.w3c.dom.Attr;
11 import org.w3c.dom.Document;
12 import org.w3c.dom.Element;
13 public class CrearXml {
14     public static void main(String argv[]) {
15         try {
16             DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
17             DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
18             //Elemento raíz
19             Document doc = docBuilder.newDocument();
20             Element rootElement = doc.createElement("root");
21             doc.appendChild(rootElement);
22             //Primer elemento
23             Element elemento1 = doc.createElement("elemento1");
24             rootElement.appendChild(elemento1);
25             //Se agrega un atributo al nodo elemento y su valor
26             Attr attr = doc.createAttribute("id");
27             attr.setValue("valor del atributo");
28             elemento1.setAttributeNode(attr);
29             Element elemento2 = doc.createElement("elemento2");
30             elemento2.setTextContent("Contenido del elemento 2");
31             rootElement.appendChild(elemento2);
32             //Se escribe el contenido del XML en un archivo
33             TransformerFactory transformerFactory = TransformerFactory.newInstance();
34             Transformer transformer = transformerFactory.newTransformer();
35             DOMSource source = new DOMSource(doc);
36             StreamResult result = new StreamResult(new File("/prueba/xmlfile.xml"));
37             transformer.transform(source, result);
38         } catch (ParserConfigurationException pce) {
39             pce.printStackTrace();
40         } catch (TransformerException tfe) {
41             tfe.printStackTrace();
42         }
43     }
44 }

```

si abrimos el documento Xml generado:



xmlfile: Bloc de notas

Archivo Edición Formato Ver Ayuda

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?><root><elemento1
id="valor del atributo"/><elemento2>Contenido del elemento
2</elemento2></root>

```

1.5.2. Lectura de documento XML

Para leer un documento XML, debemos importar las mismas librerías del ejercicio anterior, pero veamos mejor un ejemplo. tenemos un fichero XML con los siguientes datos:

```

<?xml version="1.0"?>
<company>
  <employee id="1001">

```

```

    <firstname>Tony</firstname>
    <lastname>Black</lastname>
    <salary>100000</salary>
</employee>
<employee id="2001">
    <firstname>Amy</firstname>
    <lastname>Green</lastname>
    <salary>200000</salary>
</employee>
</compan

```

Denominado xmldata.xml que se encuentra en la carpeta prueba, creamos una clase denominado SimpleTesting y el código quedará de esta manera:

```

1 import java.io.File;
2 import java.io.IOException;
3 import javax.xml.parsers.DocumentBuilder;
4 import javax.xml.parsers.DocumentBuilderFactory;
5 import javax.xml.parsers.ParserConfigurationException;
6 import org.w3c.dom.Document;
7 import org.w3c.dom.Element;
8 import org.w3c.dom.Node;
9 import org.w3c.dom.NodeList;
10 import org.xml.sax.SAXException;
11
12 public class SimpleTesting
13 {
14     public static void main(String[] args) throws ParserConfigurationException, SAXException
15     {
16         try {
17             File file = new File("c:/prueba/xmldata.xml");
18             DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
19             DocumentBuilder db = dbf.newDocumentBuilder();
20             Document document = db.parse(file);
21             document.getDocumentElement().normalize();
22             System.out.println("Root Element : " + document.getDocumentElement().getNodeName());
23             NodeList nList = document.getElementsByTagName("employee");
24             System.out.println("-----");
25             for (int temp = 0; temp < nList.getLength(); temp++) {
26                 Node nNode = nList.item(temp);
27                 System.out.println("\nCurrent Element : " + nNode.getNodeName());
28                 if (nNode.getNodeType() == Node.ELEMENT_NODE) {
29                     Element eElement = (Element) nNode;
30                     System.out.println("Employee id : " + eElement.getAttribute("id"));
31                     System.out.println("First Name : " + eElement.getElementsByTagName("firstname").item(0).getTextContent());
32                     System.out.println("Last Name : " + eElement.getElementsByTagName("lastname").item(0).getTextContent());
33                     System.out.println("Salary : " + eElement.getElementsByTagName("salary").item(0).getTextContent());
34                 }
35             }
36         } catch (IOException e) {
37             System.out.println(e);
38         }
39     }
40 }
41

```

Este sería el resultado:

```

Root Element :company
-----

Current Element :employee
Employee id : 1001
First Name : Tony
Last Name : Black
Salary : 100000

Current Element :employee
Employee id : 2001
First Name : Amy
Last Name : Green
Salary : 200000

```

1.5.3. Vinculación (Binding)

Definición

Se refiere al proceso de representación de la información en un documento XML como un objeto de negocio en la memoria del ordenador. Esto permite a las aplicaciones acceder a los datos en el XML desde el objeto en lugar de usar **DOM o SAX** para recuperar los datos desde una representación directa del XML en sí.

Esto se logra mediante la **creación automática** de una relación entre los elementos del esquema XML del documento que queremos enlazar y los miembros de una clase a ser representados en la memoria.

Cuando se aplica este proceso para convertir un documento XML a un objeto, se llama **unmarshalling**. El proceso inverso, serializar un objeto como XML, se llama **marshalling**.

Importante

XML es secuencial y los objetos en general no, las asignaciones de enlace de datos XML suelen tener ciertas dificultades para conservar toda la información de un documento XML. En concreto, información como comentarios, referencias a entidades XML y el orden hermano pueden no ser conservados en la representación de objetos creada por la aplicación enlazadora. Este no es siempre el caso; los enlazadores de datos suficientemente complejos son capaces de preservar el 100% de la información de un documento XML.

1.6. LIBRERÍAS PARA CONVERSIÓN DE DOCUMENTOS XML A OTROS FORMATOS

En este punto vamos a ver una librería que nos permite la conversión de documentos XML a otros formatos.

Xalan es un librería destinada a la transformación de textos de XML con lenguajes basados en XSL(XSLT, XPath). El XML puede ser transformado en distintos formatos, como HTML u otro documento XML. Para realizar las transformaciones de documentos XML, Xalan funciona como un procesador de plantillas XSLT.



Logo Xalan

Además es compatible con JAXP, implementando la rama correspondiente a las transformaciones XSLT de dicha API. Es multiplataforma y funciona tanto en Linux como en Windows

La herramienta nos ofrece dos maneras de realizar transformaciones XSL:

- La transformación se realiza en base a la **interpretación y procesamiento de plantillas XSLT**.
- Primeramente se **procesa la plantilla XSL** creando un conjunto de clases Java. Después estas clases son aplicadas a un documento XML para realizar la transformación.

Ventajas:

- Es **multiplataforma**, funciona tanto en linux como windows
- Es **compatible con JAXP**, implementando la rama correspondiente a las transformaciones XSLT de dicha API.
- **La salida no tiene por qué ser HTML** para visualización en un navegador, sino que puede estar en muchos formatos.
- Permite **manipular, de muy diversas maneras**, un documento XML: reordenar elementos, filtrar, añadir, borrar, etc.
- Permite **acceder a todo el documento XML**, no sólo al contenido de los elementos.
- **XSLT es un lenguaje XML**, por lo que no hay que aprender nada especial acerca de su sintaxis.

Desventajas:

- Su utilización es **más compleja**.
- **Consume cierta memoria** y capacidad de proceso, pues se construye un árbol con el contenido del documento.

Recurso web

En este enlace puedes ver algunos ejemplos

<https://xml.apache.org/xalan-j/samples.html>

Bibliografía

El gran libro de Java a Fondo 4ª 2020 de Pablo Augusto Sznajdleder (Autor)

Java 9 2018 de Herbert Schildt (Autor)

Java 9 (MANUALES IMPRESCINDIBLES) 2017 de F. Javier Moldes (Autor)